

---

# UDACITY MACHINE LEARNING NANODEGREE (MLND)

---

Capstone Project Report

Customer Segmentation – Arvato Financial Solutions

April 10<sup>th</sup>, 2019



# Project Overview

This project was one of the proposed capstone options for the Udacity Machine Learning Nanodegree. The goal for the project is to determine how one of Arvato's clients can acquire new customers for their mail-order organic products.

To help with this problem statement Arvato provides data on general population demographics, on their customers and on client response to previous campaign. This data is protected under terms and conditions and not shareable.

Using these datasets, we are proposed to predict which characteristics from individuals from the general population can be used to selectively target as good responders to this marketing campaign.

The project is broken down into 2 major subsections:

- ✚ Unsupervised Learning to identify segments of German population that match the existing customer segments;
- ✚ Supervised Learning to identify the likelihood of customer conversion from the general population

Completion of this project requires:

1. Creation of a Customers Segmentation model
2. Creation of a Supervised Learning model to qualify the performance of the predictions
3. Kaggle submission of the results obtained

All the supporting analysis and documentation (with the exception of the datasets) is available at [Github](#).

## Domain Background

Bertelsmann found its origins as a publishing house in 1835 (Schuler, 2010), and through steady growth and development made its way to the software and hardware distribution market in the 80's (Computerwoche, 1983). By 1999 the company received its current name Arvato Bertelsmann (Name, 1999) and over the next decade fully entered the domain of high-tech, information technology, and e-commerce services (Paperlein, 2012).

Arvato offers financial solutions in the form of diverse segments, from payment processing to risk management activities. It is in this domain that that this capstone project will be developed. Arvato is looking to use its available datasets to support a client (mail-order company selling organic products) in identifying the best data founded way to acquire new client base. To achieve this goal, I will explore Arvato's existing datasets to identify attributes and demographic features that can help segment customers of interest for this particular client.

Customer centric marketing is a growing field that benefits greatly from accurate segmentation, with the help of machine learning hidden patterns can be found in volumes that could easily be missed without computational help, requiring very little maintenance or human intervention, leading to an improved experience from customer seekers and customers alike.

## Problem Statement

The problem statement for this project is “How can a client – mail order company selling organic products – acquire new clients in a more efficient way?”.

The solution I propose for this problem is divided in 3 subproblems.

I will use an unsupervised learning approach to identify customer segments of value based on demographics data of existing customers versus general population data, and will follow-up on the discovered customer segments with a supervised learning approach using a dataset with demographics information for the target customers for the advertising campaign and predict which individuals would be more likely to convert to company customers.

## Datasets and Inputs

All the datasets were provided by Arvato in the context of the Udacity Machine Learning Engineer Nanodegree, on the subject of Customer Acquisition / Targeted Advertising prediction models.

There are 4 datasets to be explored in this project:

- ✚ Udacity\_AZDIAS\_052018.csv: Demographics data for the general population of Germany; 891 211 persons (rows) x 366 features (columns)
- ✚ Udacity\_CUSTOMERS\_052018.csv: Demographics data for customers of a mail-order company; 191 652 persons (rows) x 369 features (columns)
- ✚ Udacity\_MAILOUT\_052018\_TRAIN.csv: Demographics data for individuals who were targets of a marketing campaign; 42 982 persons (rows) x 367 (columns).
- ✚ Udacity\_MAILOUT\_052018\_TEST.csv: Demographics data for individuals who were targets of a marketing campaign; 42 833 persons (rows) x 366 (columns).

And 2 metadata files associated with these datasets:

- ✚ DIAS Information Levels — Attributes 2017.xlsx: a top-level list of attributes and descriptions, organized by informational category
- ✚ DIAS Attributes — Values 2017.xlsx: a detailed mapping of data values for each feature in alphabetical order

Which can help mapping the attributes to its type or missing value encoding.

## Evaluation Metrics

This problem is a multi-class classification problem, and one the most valuable metrics to measure model performance is the Area Under the Curve Receiver Operating Characteristics (ROC-AUC). The curve represents a degree or measure of separability and, the higher the score the better the model is performing.

A great advantage of using ROC-AUC is the immunity to class imbalance, which is the case for this problem. The number of people that are positive responders to an ad campaign are on average far lesser than those that respond negatively.

This is also the required evaluation metric for the [Kaggle](#) submission.

## EDA and Preprocessing

	LNR	AGER_TYP	AKT_DAT_KL	ALTER_HH	ALTER_KIND1	ALTER_KIND2	ALTER_KIND3	ALTER_KIND4	ALTERSKATEGORIE_FEIN
0	910215	-1	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	910220	-1	9.0	0.0	NaN	NaN	NaN	NaN	21.0
2	910225	-1	9.0	17.0	NaN	NaN	NaN	NaN	17.0
3	910226	2	1.0	13.0	NaN	NaN	NaN	NaN	13.0
4	910241	-1	1.0	20.0	NaN	NaN	NaN	NaN	14.0
...	...	...	...	...	...	...	...	...	...
891216	825761	-1	5.0	17.0	NaN	NaN	NaN	NaN	17.0
891217	825771	-1	9.0	16.0	NaN	NaN	NaN	NaN	16.0
891218	825772	-1	1.0	17.0	NaN	NaN	NaN	NaN	17.0
891219	825776	-1	9.0	0.0	17.0	NaN	NaN	NaN	20.0
891220	825787	-1	1.0	0.0	NaN	NaN	NaN	NaN	NaN

891221 rows × 366 columns

Each line of the demographic data file represents a person, but also contains information other than the individual, including information about their home, building, and neighbors.

## Data Preprocessing

After working on some projects, I've to say that I understand what people mean when they say that most of the time (up to 80 %) are spent on ETL, EDA, and other such tasks. This project wasn't an exception. The steps are as follows:

### Step 1: Create a complete feature dictionary file(feat\_info.csv)

There are only 314 out of 366 items in the dictionary, meaning that there are a lot of features that are not listed in the given attribute description file. So, the first thing I did was to create a complete feature dictionary file, which containing all the 366 features in our datasets.

### Step 2: Convert Unknown and Missing Values to NaN

	index	Attribute	Description	Value	Meaning
0	1	AGER_TYP	best-ager typology	-1	unknown
5	6	ALTERSKATEGORIE_GROB	age classification through prename analysis	-1, 0	unknown
33	34	ANREDE_KZ	gender	-1, 0	unknown
40	41	BALLRAUM	distance to next urban centre	-1	unknown
48	49	BIP_FLAG	business-flag indicating companies in the buil...	-1	unknown

The third column unknown of the feature dictionary file indicate missing or unknown data. We convert data that matches a 'missing' or 'unknown' value code into a NaN values. The dataframe below shows how much data is NaN in top ten columns.

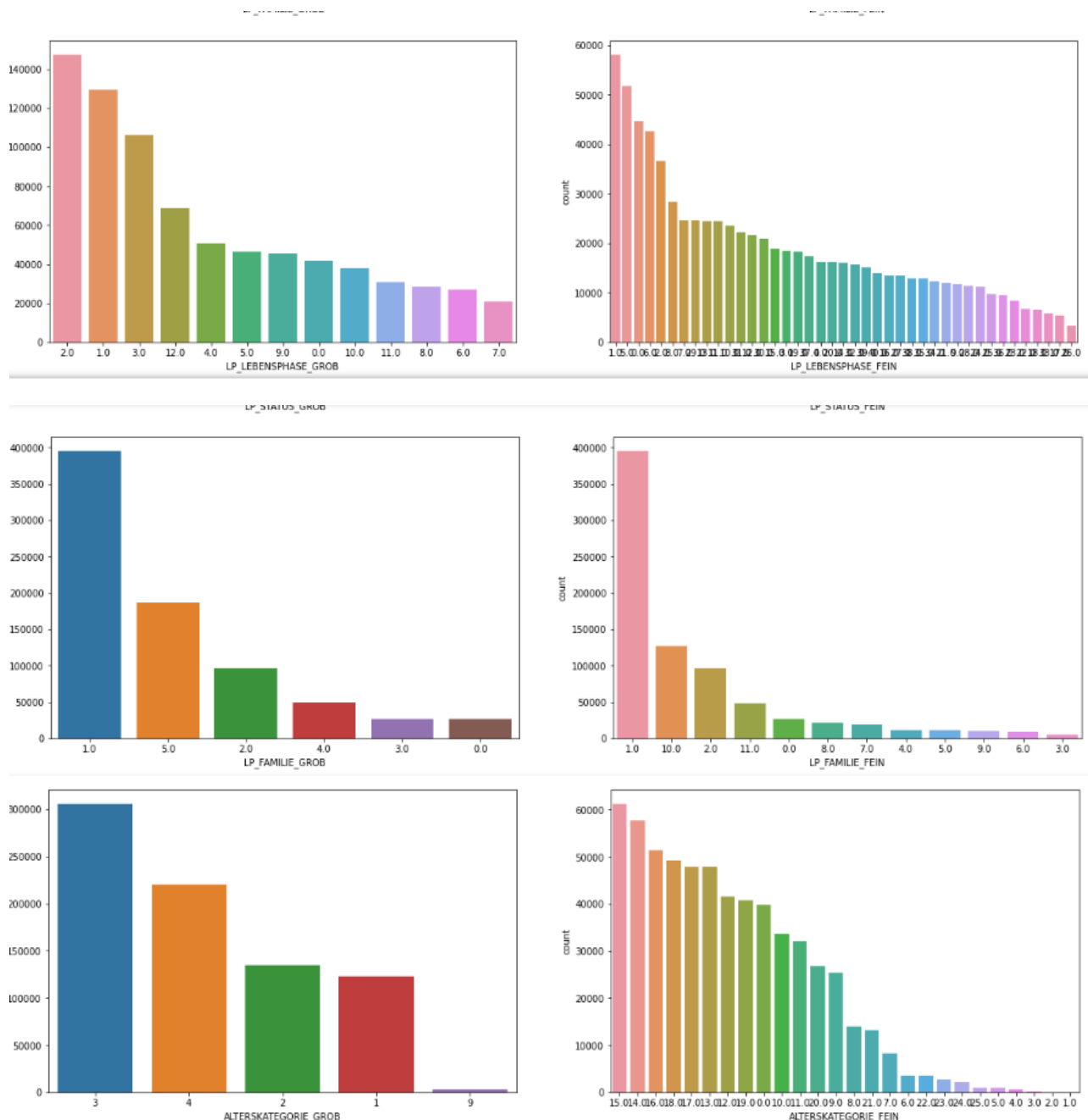
	Total	Percent
ALTER_KIND4	890016	0.998648
TITEL_KZ	889061	0.997576
ALTER_KIND3	885051	0.993077
ALTER_KIND2	861722	0.966900
ALTER_KIND1	810163	0.909048
AGER_TYP	677503	0.760196
EXTSEL992	654153	0.733996
KK_KUNDENTYP	584612	0.655967
KBA05_BAUMAX	476524	0.534687
ALTERSKATEGORIE_FEIN	262947	0.295041

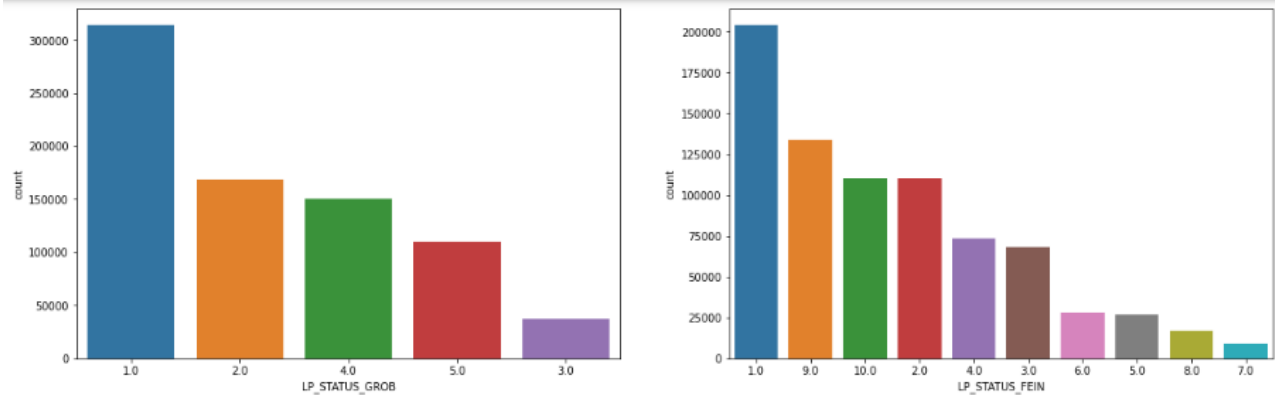
### Step 3: Remove Unknown and Missing Values

I decide to drop all the features that had more than 30% NaN values, in our case 58 features. Also I drop all the rows that had more than 25% NaN values.

### Step 4: Remove duplicated features

There are a lot of the features are redundant, For instance, The suffix \_FEIN indicates fine-grained information, while the suffix \_GROB indicates coarse-grained information.





I decided to remove these duplicate features, 'ALTERSKATEGORIE\_GROB', 'LP\_FAMILIE\_GROB', 'LP\_LEBENSphase\_FEIN', 'LP\_STATUS\_GROB', 'CAMEO\_DE U\_2015'.

Because some of them are not defined, or the corresponding GROB feature is enough.

## Step 5: Re-encodings features

There are many features that need to be re-encoded. For example, two of the values of the OST\_WEST\_KZ feature are the characters O, W. Since machine learning models techniques only respond to numerical data, we need to re-encode it.

```
#investigating 'OST_WEST_KZ' values
azdias['OST_WEST_KZ'].value_counts(dropna=False)
```

```
W    619190
O    166112
Name: OST_WEST_KZ, dtype: int64
```

```
#check values after re-encoding
azdias['OST_WEST_KZ'].value_counts()
```

```
1    619190
0    166112
Name: OST_WEST_KZ, dtype: int64
```

Also, noticed there are features with only values '1' and '2' so I decided to re-encode them as binary '0' and '1'

```
#investigating binary columns ['VERS_TYP']
azdias['VERS_TYP'].value_counts()
```

```
2.0    389009
1.0    360215
Name: VERS_TYP, dtype: int64
```

```
# investigating binary column ['ANREDE_KZ']
azdias['ANREDE_KZ'].value_counts()
```

```
2    409631
1    375671
Name: ANREDE_KZ, dtype: int64
```

```
#standardizing binary into 0,1
azdias['VERS_TYP'].replace([2.0, 1.0], [1, 0], inplace=True)
azdias['ANREDE_KZ'].replace([2, 1], [1, 0], inplace=True)
```

```
#checking results
azdias['VERS_TYP'].value_counts()
```

```
1.0    389009
0.0    360215
Name: VERS_TYP, dtype: int64
```

```
azdias['ANREDE_KZ'].value_counts()
```

```
1    409631
0    375671
Name: ANREDE_KZ, dtype: int64
```

Finally, as a last step in the preprocessing portion I performed some Feature Encoding and Engineering.

'CAMEO\_DEU\_2015' is a categorical feature that ranged from 1 to 9 and A to F I used a `LabelEncoder()` to encode the categories to ints.

,

'EINGEFUEGT\_AM' is a time related feature, so I converted it to a datetime object and extracted the year I created 2 different features from 'PRAEGENDE\_JUGENDJAHRE', a 'GENERATION' feature and a type of 'MOVEMENT' feature (avant-garde or not)

'LP\_LEBENSPHASE\_FEIN' was used to create two new features, one related to life stage, 'LP\_LEBENSPHASE\_FEIN\_STAGE' and one related to the wealth scale 'LP\_LEBENSPHASE\_FEIN\_SCALE'

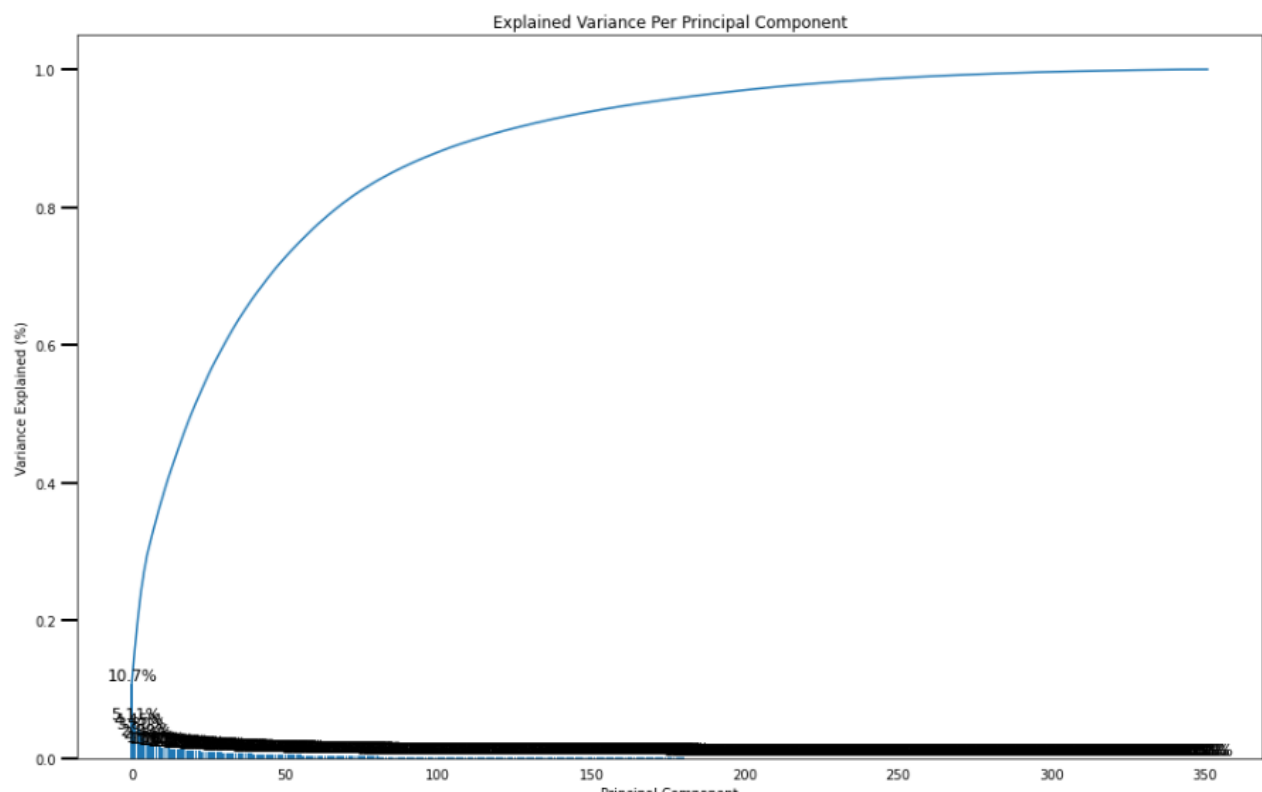
Once I had created and encoded all these features I used `SimpleImputer()` to impute the nans with the most-frequent values.



## Dimensionality Reduction

It was said by someone else much better than I probably can: “As the number of features increase, the number of samples also increases proportionally. The more features we have, the greater number of samples we will need to have all combinations of feature values well represented in our sample.” (Raj, 2019) and this can make datasets incredibly complex and prone to issues like overfitting (a model becomes so complex that it becomes hard to generalize it to anything else). Dimensionality reduction leads to the creation of models that are more accurate due to the reduction of misleading data, it leads to the need of less computing power and storage and a reduction in feature noise.

I used **robust scaler** to scale my data. Once I had my data scaled, I used principal component analysis (PCA) which is a method of dimensionality reduction through feature extraction. To decide how many components, I should keep that accounted for over 85% of the variance observed in the results, I used a scree plot (line plot of the eigenvalues of factors or principal components in an analysis).

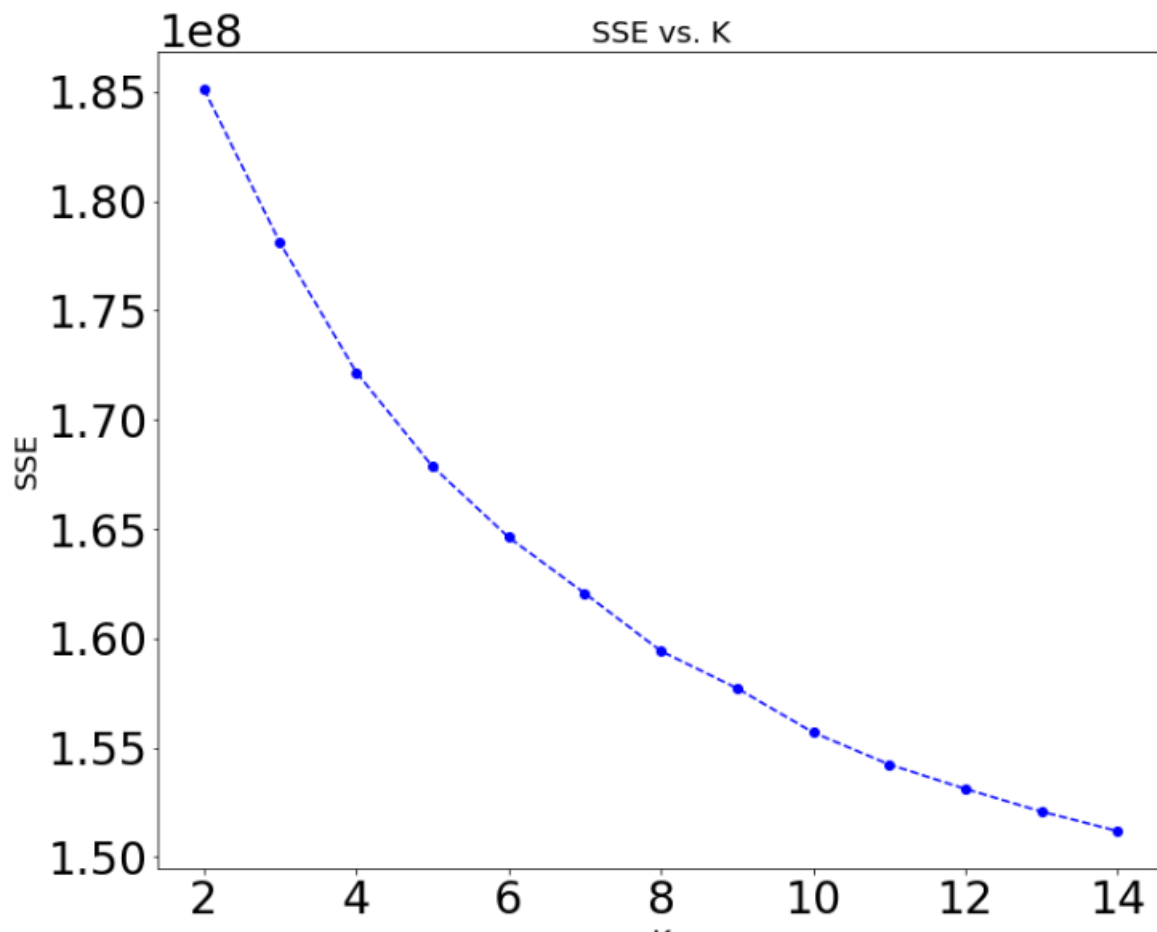


using **robust scaler** with 90 principal components 86% of the original variance can be represented

# Customer Segmentation Report

Clustering groups data based on similarity. After choosing the principal components, we hope to help segment customers by aggregating these principal components. I used both scikit-learn's K-Means and GMM to cluster the principal components and compare between them .

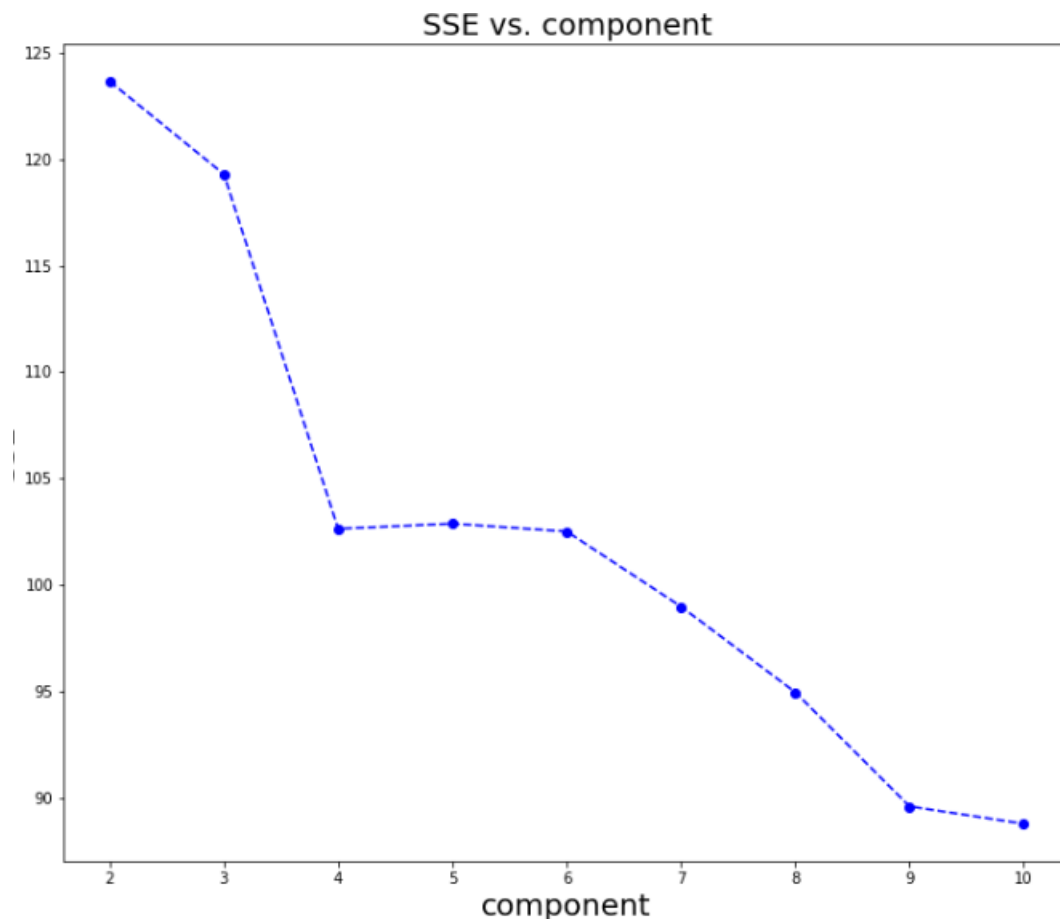
One of the first steps with K-means is to define a k (number of centroids) that are the imaginary centers for each cluster. To help me select an optimal k I will use the Elbow method. The Elbow method compares the within cluster sum of squares until there is no more visible improvement. This is quite visually striking when using plots:



Based on the Elbow method we know that for these datasets and the data cleaning I implemented k = 8 seems to be the ideal.

One of the first steps with GMM is to define a number of components (number of centroids) that are the

imaginary centers for each cluster. To help me select an optimal number of components I will use the Elbow method. The Elbow method compares the within cluster sum of squares until there is no more visible improvement. This is quite visually striking when using plots:



As shown above , **Sum of Squared Error(SSE)** in the **GMM** model is much lower than of the **K-means** model ,However, I decided to take it one step further and use **silhouette score** to compare the two models and I have chosen number of components in the **GMM** to be **9** based on **ELBOW METHOD**

```
#predicting clusters
gmm_azdias = gmm_model.predict(azdias_pca)
```

```
sil_Score_gmm_az= silhouette_score(azdias_pca, gmm_azdias)
sil_Score_gmm_az
```

```
-0.012259899793620615
```

```
#creating silhouette_score and fitting it to predicted kmeans model
sil_score_km = silhouette_score(azdias_pca, km_azdias)
```

```
sil_score_km
```

```
0.03296471132873044
```

```
#creating silhouette_score and fitting it to predicted kmeans model
sil_score_km = silhouette_score(azdias_pca, km_azdias)
```

```
sil_score_km
```

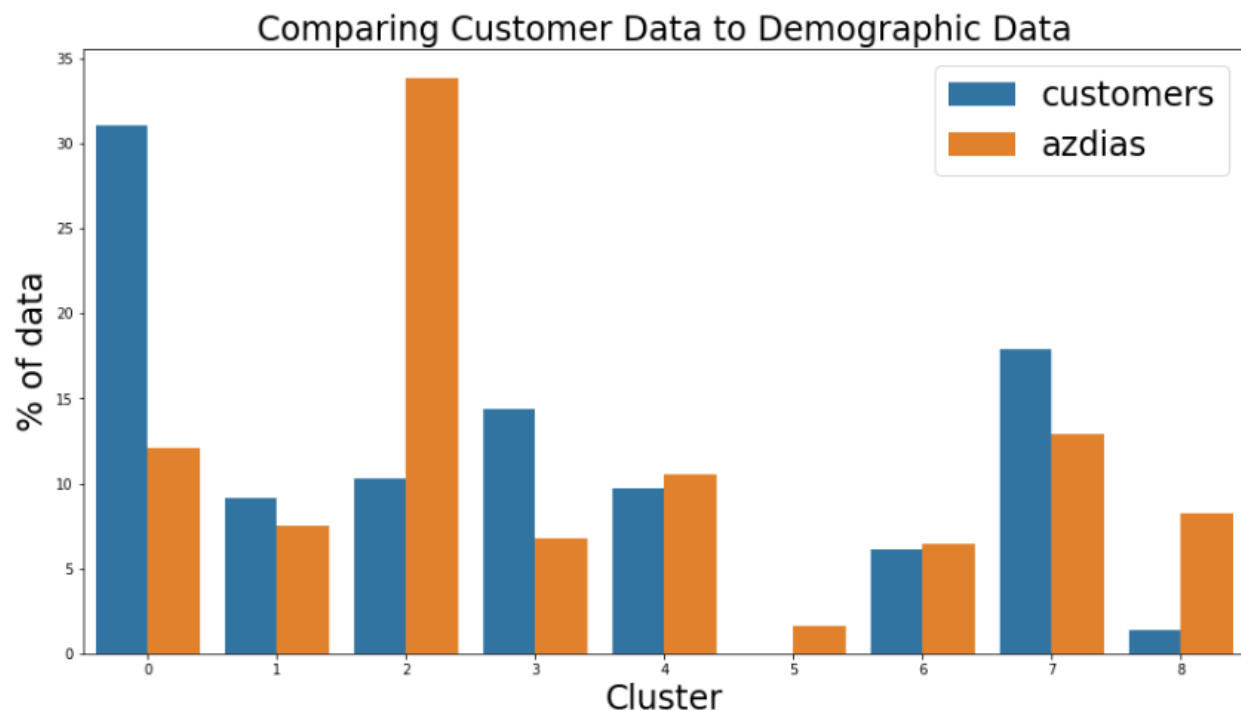
```
0.03296471132873044
```

```
#predicting clusters
gmm_azdias = gmm_model.predict(azdias_pca)
```

```
sil_Score_gmm_az= silhouette_score(azdias_pca, gmm_azdias)
sil_Score_gmm_az
```

```
-0.012259899793620615
```

As shown , difference in silhouette score is around 0.04 which is negligible , while it was possible to run a silhouette score analysis for number of K in K-means model and number of components in GMM model to perhaps obtain a better model with a more optimum silhouette score , it was unreasonable computationally extensive , so I decided against it and to be content with optimizing SSE only



Cluster 0,3 are overrepresented, Cluster 2, 8, 5 are underrepresented.

## Supervised Learning Model

We are reaching the closing portion, and for me my favorite part (because I can compare my results to others and have an idea of how well my work performed).

Comparing the positive vs negative responses truly shows how imbalanced the data is:





```
train['RESPONSE'].value_counts()
```

```
0    42430
1      532
Name: RESPONSE, dtype: int64
```

```
#calculating reponse rate
responded_percentage = train['RESPONSE'].value_counts()[1]/train['RESPONSE'].value_counts()[0] *100
print('customers responded =',responded_percentage , '%')
print('customers ignored =', 100 - responded_percentage , '%')
```

```
customers responded = 1.253829837379213 %
customers ignored = 98.74617016262079 %
```

There were quite a few classifiers I wanted to try for this portion of the project:

-  RandomForestClassifier
-  XGBClassifier
-  LGBMClassifier
-  GradientBoostingClassifier

In principle there was a possibility of any of these to work for this problem, but I decided to let the data speak its truth and pit them against each other

```
rf = RandomForestRegressor(random_state=42)
model_grid(rf,{})
```

0.7048612968021268

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                        max_depth=None, max_features='auto', max_leaf_nodes=None,
                        max_samples=None, min_impurity_decrease=0.0,
                        min_impurity_split=None, min_samples_leaf=1,
                        min_samples_split=2, min_weight_fraction_leaf=0.0,
                        n_estimators=100, n_jobs=None, oob_score=False,
                        random_state=42, verbose=0, warm_start=False)
```

```
xgb = XGBRegressor(objective='binary:logistic', random_state=42)
model_grid(xgb,{})
```

0.7572387214697314

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              importance_type='gain', learning_rate=0.1, max_delta_step=0,
              max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
              n_jobs=1, nthread=None, objective='binary:logistic',
              random_state=42, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
              seed=None, silent=None, subsample=1, verbosity=1)
```

```
gb = GradientBoostingRegressor(random_state=42)
model_grid(gb,{})
```

0.7535573144172779

```
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                            init=None, learning_rate=0.1, loss='ls', max_depth=3,
                            max_features=None, max_leaf_nodes=None,
                            min_impurity_decrease=0.0, min_impurity_split=None,
                            min_samples_leaf=1, min_samples_split=2,
                            min_weight_fraction_leaf=0.0, n_estimators=100,
                            n_iter_no_change=None, presort='deprecated',
                            random_state=42, subsample=1.0, tol=0.0001,
                            validation_fraction=0.1, verbose=0, warm_start=False)
```

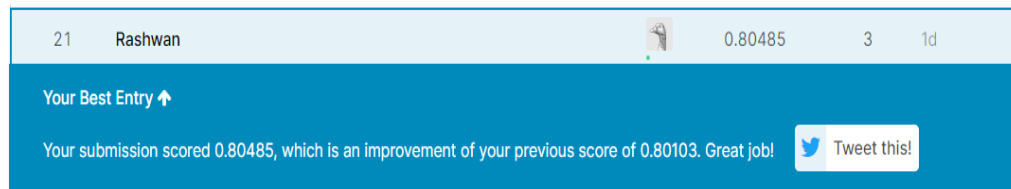
```
lgbc = lgb.LGBMClassifier(application='binary', random_state=42)
model_grid(lgbc,{})
```

0.7011015726712928

Based on the model comparisons it became clear that there were 3 winners. Gradient Boost, XGB and LGBM, so out of these 3 I selected LGBM and XGB for further optimization and testing.

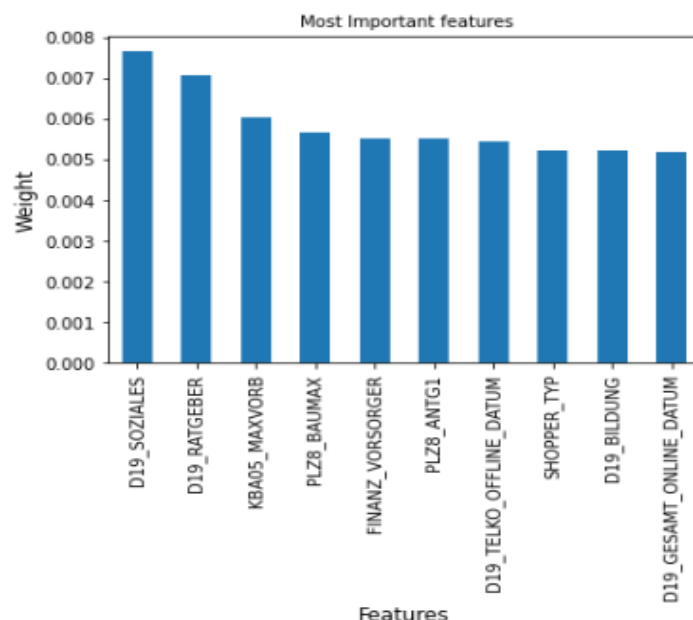
Out of all the available automated tuning approaches I chose a Randomized optimization approach since being the least time consuming.

# Results



My model landed me on the first quartile (21 out of 144 currently) of the Kaggle leaderboard, which is really exciting, it means that there is room for improvement (patience with running xgboost for more iterations, using techniques to deal with the class imbalance like SMOTE, play a little bit more with my data cleaning techniques).

I also identified D19\_SOZIALES as the feature with the greatest impact in the model.



## Improvements and continued work

This project was incredible, challenging and gratifying, I am excited to see if I have room to improvement with my approach. In the future I plan to manipulate the following segments of my approach to this problem:

- ✚ applying a silhouette score analysis to choose a good K for K means or potentially a good number of components for GMM
- ✚ Apply sampling techniques to balance classes
- ✚ Run BayesSearchCV with different scaling models

Thank You



## Works Cited

- Brems, M. (2017, April 17). A One-Stop Shop for Principal Component Analysis. *Towards Data Science*.
- Computerwoche. (1983, October 21). Bertelsmann vertreibt Rechner von TI. *Computerwoche*.
- Ding, C. a. (2004). K-Means Clustering via Principal Component Analysis. *Proceedings of the Twenty-First International Conference on Machine Learning*.
- Garbade, M. J. (2018, September 12). Understanding K-means Clustering in Machine Learning. *Towards Data Science*.
- Gove, R. (2017, December 26). Using the elbow method to determine the optimal number of clusters for k-means clustering. *Robert Gove's Block 0060ff3b656618e9136b*.
- Koehrsen, W. (2018, July 3). Automated Machine Learning Hyperparameter Tuning in Python. *Towards Data Science*.
- Name, N. (1999, 9 June). Darmstädter Echo. *neue Ziele*.
- Paperlein, J. (2012, April 5). E-Commerce ist weltweit ein Thema. *Horizont*, p. 14.
- Raj, J. T. (2019, March 11). A beginner's guide to dimensionality reduction in Machine Learning. *Towards Data Science*.
- Schuler, T. (2010, June 18). Erst Drucker, dann Verleger. *Berliner Zeitung*, p. 30.